Modulare Spezifikation und Steuerung von Dialogen in web-basierten Anwendungen

Matthias Book, Volker Gruhn

Lehrstuhl für Angewandte Telematik/e-Business,* Universität Leipzig
Klostergasse 3, 04109 Leipzig
{book, gruhn}@ebus.informatik.uni-leipzig.de

Abstract: Die Benutzerfreundlichkeit von Web-Anwendungen leidet häufig unter der fehlenden Unterstützung hierarchischer Dialogsequenzen, an die Benutzer sich während der Arbeit mit fensterbasierten Benutzeroberflächen und ihren hierarchisch strukturierten Dialogfenstern bereits gewöhnt haben. Für Multikanal-Anwendungen liegt eine zusätzliche Herausforderung darin, die geräteunabhängige Geschäftslogik mit den gerätespezifischen Interaktionsmustern zu verbinden, die die unterschiedlichen Ein-/Ausgabefähigkeiten verschiedener Clients erforderlich machen. Um diese Probleme anzusprechen, stellen wir eine grafische Dialogfluss-Notation vor, die die Spezifikation hierarchischer Dialogsequenzen für verschiedene Präsentationskanäle ermöglicht. Nach Übersetzung in eine XML-basierte Dialogbeschreibungssprache eignet sich diese Spezifikation als Eingabe für ein Framework, das die Dialogflüsse komplexer Web-Anwendungen steuert.

1 Einleitung

Eine der zentralen Herausforderungen bei der Entwicklung von Web-Anwendungen ist der Unterschied zwischen seiten- und fensterbasierten Oberflächenparadigmen: In fensterbasierten Anwendungen kann jedes Fenster „Kind-Fenster“ öffnen, wobei der Benutzer nach Abschluss eines Dialogs zum Dialog im „Eltern-Fenster“ zurück gelangt. Benutzer können sich auf diese vorhersagbare „Eltern-Kind-Beziehung“ zwischen Fenstern verlassen, die ihr konzeptuelles Modell der Anwendung bestärkt und so die Benutzerfreundlichkeit der Anwendung steigert. Im Web muss diese Struktur hingegen durch eine entsprechende Seitenabfolge simuliert werden, da jede neue Seite die vorherige ersetzt, statt sie zu „überlagern“ und so eine einfache „Rückkehr“ zu erlauben [RFPG96].

Während einfache lineare und verzweigte Dialogsequenzen noch mit grundlegenden Zustands-Verwaltungsmechanismen implementiert werden können, erfordern Eltern-Kind-Beziehungen zwischen Seiten jedoch eine komplexere Dialogsteuerungslogik. Um diesen Aufwand zu vermeiden, bieten die meisten der heutigen Web-Anwendungen keine verschachtelten Dialogsequenzen – stattdessen müssen Benutzer manuell zwischen „Eltern-“ und „Kind-Seiten“ navigieren (wenn solch ein hierarchischer Zusammenhang überhaupt

*Der Lehrstuhl für Angewandte Telematik/e-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG.

erkennbar ist). Dies widerspricht jedoch dem konzeptuellen Modell verschachtelter Dialoge, das Benutzer während der langjährigen Arbeit mit fensterbasierten Oberflächen etabliert haben – eine Verletzung der ISO-Dialoggrundsätze der Steuerbarkeit und Erwartungskonformität [iso96], der hohe kognitive und Gedächtnisanforderungen an den Benutzer stellt und so die Benutzerfreundlichkeit der Web-Anwendungen mindert.

Während die Entwicklung individueller fensterbasierter Oberflächen für Plattformen mit verschiedenen Betriebssystem-, Speicher- und Ein-/Ausgabe-Restriktionen [JHE99] u.U. viel Aufwand erfordert, wenn die gleiche Anwendung über viele verschiedene Geräte (z.B. PCs, PDAs, Mobiltelefone, Blindenlesegeräte) zugreifbar sein soll, kann eine hypertext-basierte Oberflächenbeschreibung (ggf. syntaktisch angepasst) auf verschiedenen Präsentationskanälen angezeigt werden. Die unterschiedlichen Ein-/Ausgabefähigkeiten verschiedener Geräte beeinflussen jedoch, wie Benutzer mit einer Anwendung arbeiten: Ein Dialog, der mit einem Desktop-Browser in einem einzigen Schritt abzuschließen ist, muss auf einem mobilen Gerät u.U. in mehrere Interaktionsschritte aufgebrochen werden. Die server-seitige Geschäftslogik sollte allerdings unabhängig von solchen Client-Eigenheiten bleiben [BGGW02]. Dies erfordert offensichtlich eine logisch getrennte Implementierung von Darstellungs- und Geschäftslogik – wie wir in Abschnitt 3 sehen werden, ist dies aber nicht so trivial, wie es klingt, da die Dialogsteuerungslogik oft mit den beiden anderen Logikschichten vermischt wird.

Um die Probleme der hierarchischen Dialoge und gerätespezifischen Interaktionsmuster anzusprechen, stellen wir eine Dialogfluss-Notation vor, die die Spezifikation komplexer Dialogflüsse erlaubt (Abschnitt 2) und präsentieren ein Dialogsteuerungs-Framework, das die korrespondierende Dialogsteuerungslogik zur Wiederverwendung in beliebigen Anwendungen bereitstellt (Abschnitt 3).

2 Dialogfluss-Notation

Die Dialogfluss-Notation *(Dialog Flow Notation, DFN)* stellt den Dialogfluss einer Anwendung als gerichteten (jedoch nicht notwendigerweise bipartiten) Graphen verschiedener Arten von Zuständen dar, die durch Transitionen verbunden sind (ein sog. *Dialoggraph*). Einige Konstrukte und Symbole der DFN wurden Harels Statecharts [Har87] entlehnt, haben jedoch eine dialogflussspezifische Semantik. Die DFN bezeichnet Transitionen als *Ereignisse* und Zustände als *Dialogelemente*, wobei atomare und zusammengesetzte Dialogelemente unterschieden werden.

Wir unterscheiden zwei Arten von atomaren Dialogelementen: *Masken* (symbolisiert durch Quadrate mit „Eselsohren“) bezeichnen auf dem Client dargestellte Hypertext-Seiten, während *Aktionen* (symbolisiert durch Kreise) auf dem Server ausgeführte Anwendungslogik repräsentieren. Ereignisse können Parameter tragen, die anwendungsspezifische Daten wie z.B. in ein Formular eingegebene Informationen oder von der Anwendungslogik berechnete Ergebnisse enthalten und so zum Datenaustausch zwischen Masken und Aktionen dienen. Semantisch verhalten Ereignisse sich nicht wie Aufrufe von Folgeelementen, sondern wie Ergebnisse der sie erzeugenden Dialogelemente, die in Abhängigkeit vom Kon-

text und generierenden Element zu verschiedenen Zielen führen können. Durch die Verknüpfung von Masken, Aktionen und Ereignissen zu Dialoggraphen können Entwickler die Interaktionsmöglichkeiten des Benutzers mit der Anwendung spezifizieren.

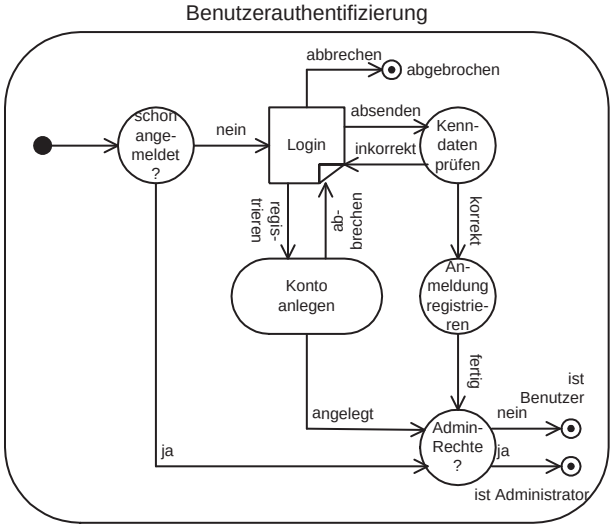

Abbildung 1: Dialoggraph des Moduls „Benutzerauthentifizierung“

Theoretisch ließe sich der gesamte Dialogfluss einer Anwendung nur mit atomaren Dialogelementen beschreiben. Eine solche Spezifikation wäre jedoch für praktische Zwecke viel zu unübersichtlich und unterstützt keine Wiederverwendung häufig vorkommender Muster innerhalb des gleichen Dialogflusses. Die DFN stellt daher sog. *Dialogmodule* bereit, die Dialogfluss-Teile einkapseln und so die Spezifikation verschachtelter Dialogstrukturen erlauben: Der innere Dialoggraph eines Moduls kann Submodule enthalten und selbst in den Dialoggraph eines Supermoduls eingebunden sei (ein Submodul kann dabei in mehrere verschiedene Dialogmodule eingebunden werden). Wenn ein Submodul aus einem äußeren Dialoggraphen aufgerufen wird, in den es eingebettet ist, so beginnt die Traversierung seines inneren Dialoggraphen stets mit dem *Initialereignis* (symbolisiert durch einen großen Punkt). Bei der Terminierung des inneren Dialoggraphen wird ein *Terminalereignis* generiert (symbolisiert durch einen umrandeten kleinen Punkt), das an das aufrufende Dialogmodul übergeben wird und dort die Traversierung des Dialoggraphs fortsetzt.

Das Modul „Benutzerauthentifizierung“ in Abb. 1 illustriert die Eigenschaften von Dialogmodulen: Dieses Modul prüft, ob der Benutzer bereits angemeldet ist und zeigt bei Bedarf die „Login“-Maske an, um Benutzername und Passwort abzufragen. Wenn die Kenndaten des Benutzers korrekt sind, kennzeichnet das Model ihn als angemeldet, prüft seine Zugangsrechte und terminiert, wobei das aufrufende Modul von den Rechten des Benutzers in Kenntnis gesetzt wird (in diesem Modul kann der Dialogfluss nun entsprechend dieses Ergebnisses fortgesetzt werden, z.B. durch Anzeige einer Seite mit den Daten, die die Authentifizierung erforderten). Falls der Benutzer noch kein Konto hat, kann er sich mit

dem eingeschachtelten Submodul „Konto anlegen“ registrieren. Indem die Geschäftslogik in feingranulare Aktionen aufgeteilt wurde, anstatt sie in einem umfassenden Block zu implementieren, kann das Modul flexibel auf verschiedene Situationen reagieren und z.B. die Kenndatenabfrage überspringen, wenn der Benutzer bereits angemeldet ist.

Die DFN stellt eine Reihe weiterer Ereignistypen und Dialogstrukturen zur Verfügung [BG03], die wir aus Platzgründen nicht näher vorstellen.

Um unterschiedliche Interaktionsmuster zu spezifizieren, die für unterschiedliche Endgeräte benötigt werden, können in der DFN mehrere Varianten eines Dialogmoduls mit individuellen Dialogflüssen für verschiedene Präsentationskanäle spezifiziert werden. Der Name des jeweiligen Präsentationskanals wird dabei als sog. *Kanalkennung* in eckigen Klammern nach dem Namen des Moduls notiert. Abb. 2 spezifiziert z.B. die Dialoggraphen des Bestellmoduls eines Online-Shops für den HTML- und WML-Präsentationskanal. Während die Kanäle verschiedene Dialogmasken verwenden, um auf die unterschiedlichen Ein-/Ausgabefähigkeiten der Endgeräte einzugehen, nutzen sie die gleichen Aktionen, um die Benutzereingaben zu verarbeiten. Dies erlaubt Entwicklern, die geräteunabhängige Anwendungslogik nur einmal zu entwickeln und dann auf mehreren Präsentationskanälen wieder zu verwenden.

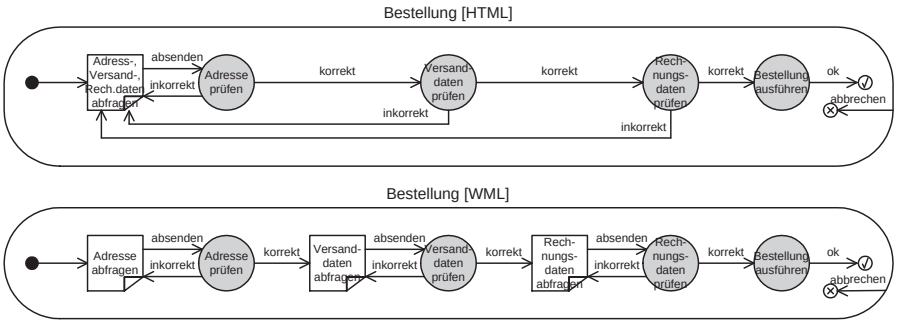

Abbildung 2: Dialoggraphen des Modules „Bestellung“ im HTML- und WML-Präsentationskanal

Zu beachten ist, dass die Adaption für verschiedene Interaktionsmuster umso einfacher ist, je feingranularer die Aktionen realisiert sind. Sehr hohe Granularität erzeugt jedoch auch hohen Spezifikations-, Implementierungs- und Performance-Overhead, sodass der Entwickler eine Balance zwischen der gewünschten Flexibilität und der erforderlich Granularität finden muss. Lösungen dieses Dilemmas und Automatisierungsmöglichkeiten für diesen derzeit noch manuell durchzuführenden Designschritt sind eine zentrale Frage unserer laufenden Forschung auf diesem Gebiet.

Nachdem der Dialogfluss einer Anwendung in der DFN spezifiziert wurde, ist ein effizienter Übergang von der Spezifikation zur Implementierung wünschenswert: Die Diagramme sollten nicht nur den Dialogfluss visualisieren, seine Implementierung aber dem Entwickler überlassen, sondern vielmehr als direkte Eingabe einer anwendungsunabhängigen Dialogsteuerungslogik dienen. Zu diesem Zweck muss die grafische Spezifikation zunächst in eine maschinenlesbare Darstellung transformiert werden, die von der Dialogsteuerungs-

logik verarbeitet werden kann. Wir haben dazu die *Dialog Flow Specification Language (DFSL)* entwickelt, eine XML-basierte Sprache, deren Elemente eng mit den Dialogelementen, Ereignissen und Konstrukten der DFN korrespondieren. Zur Unterstützung des Transformationsprozesses haben wir ein Eclipse-Plugin mit einem grafischen Editor entwickelt, der ein auf der OMG Meta-Object Facility basierendes Meta-Modell der Dialogfluss-Notation verwendet, um die gezeichneten Modelle zu validieren und daraus automatisch DFSL-Dokumente zu generieren.

3 Dialogsteuerungs-Framework

Web-Anwendungen folgen üblicherweise dem Model-View-Controller-Paradigma (MVC) [Kra88], das die Trennung von Darstellungs-, Anwendungs- und Steuerungslogik empfiehlt. Während Darstellungs- und Anwendungslogik recht intuitiv unterschieden werden können („was der Benutzer sieht“ vs. „was das System tut“), ist die Differenzierung zwischen Anwendungs- und Dialogsteuerungslogik subtiler („was das System tut“ vs. „was es in Abhängigkeit vom aktuellen Zustand sowie neuen Eingaben und Berechnungen als nächstes tun soll“). Die vermischte Implementierung von Anwendungs- und Dialogsteuerungslogik passiert daher leicht, auch wenn sie von der Darstellungslogik getrennt ist.

Dies zeigt sich z.B. im populären Struts-Framework [Apa], in dem der Dialogfluss von sogenannten Actions gesteuert wird. Sie implementieren nicht nur die Anwendungslogik, sondern entscheiden auch, an welche Maske eine Anfrage weitergeleitet werden soll – der Controller führt diese Weiterleitung nur noch aus.

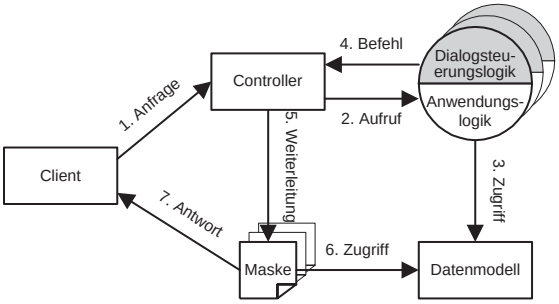

Abbildung 3: Grobarchitektur des Struts-Frameworks

Wie die Schraffierung in Abb. 3 zeigt, ist die Dialogsteuerungslogik in diesem Ansatz über alle Actions verteilt, d.h. der Dialogfluss ist nicht außerhalb der Anwendungslogik spezifiziert, sondern tatsächlich im Java-Code der Actions implementiert. Dies erlaubt den Actions nur relativ isolierte Dialogfluss-Entscheidungen zu treffen und behindert die Implementierung komplexerer Dialogstrukturen mit Konstrukten wie verschachtelten Dialogmodulen.

Um das Bewusstsein der Actions auf das Gesamtbild des Dialogflusses auszudehnen und

ihnen die Steuerung komplexerer Dialogkonstrukte zu ermöglichen, müsste noch mehr Steuerungslogik in ihnen implementiert werden, was das Problem nur noch verschärfen würde. Zudem ist die „fest verdrahtete“ dezentrale Implementierung der Dialogsteuerungslogik relativ unflexibel, zur Wiederverwendung nahezu ungeeignet und schwer zu warten. Schließlich erfordert die Realisierung von Geräteunabhängigkeit zusätzlichen und u.U. redundanten Aufwand: Da die Dialogflüsse vom Präsentationskanal abhängig sind, während die Anwendungslogik davon unabhängig sein sollte, verhindert ihre enge Kopplung die Wiederverwendung von Actions auf verschiedenen Kanälen. Stattdessen würde jeder Kanal seine eigene Menge von Actions erfordern, um den individuellen Dialogfluss des betreffenden Geräts zu implementieren.

Das Dialogsteuerungs-Framework *(Dialog Control Framework, DCF)* realisiert im Gegensatz dazu eine sehr strikte Implementierung des MVC-Patterns, in der nicht nur die Anwendungs- und Darstellungslogik, sondern auch die Dialogfluss-Spezifikation und Dialogsteuerungslogik von einander getrennt sind: Der Controller entscheidet anhand der Dialogfluss-Spezifikation, welche Aktionen und/oder Masken eingehende Anfragen bearbeiten sollen.

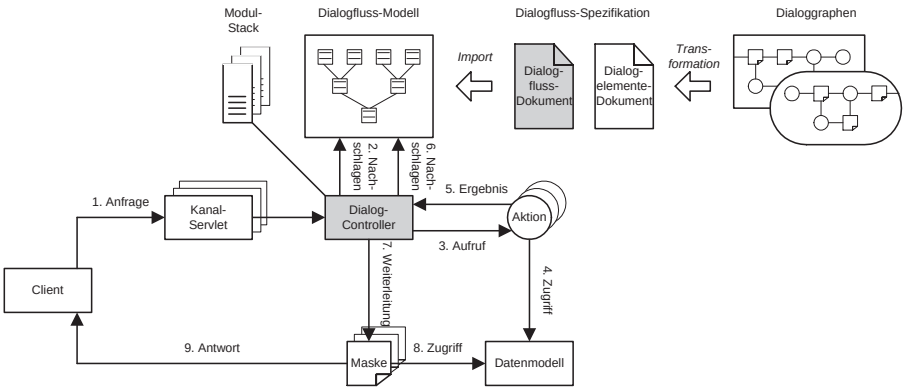

Abbildung 4: Grobarchitektur des Dialogsteuerungs-Frameworks

Wie die Grobarchitektur in Abb. 4 zeigt, sind die Aktionen hier leichtgewichtiger, da sie nur Aufrufe der Anwendungslogik enthalten, während die gesamte Dialogsteuerungslogik in den Dialog-Controller verschoben wurde. Dieser Controller empfängt keine unmittelbaren Anfragen von den Clients mehr, sondern verarbeitet Ereignisse, die von sog. *Kanal-Servlets* aus den gerätespezifischen Anfragen extrahiert wurden. Der Dialog-Controller schlägt die Empfänger dieser Ereignisse im *Dialogfluss-Modell* nach, einer Sammlung von Objekten, die Dialogelemente repräsentieren und über Referenzen miteinander verbunden sind, die die Ereignisse des Dialogflusses wiederspiegeln. Dieses Dialogfluss-Modell wird bei der Initialisierung des Frameworks aufgebaut, indem die DFSL-Dokumente mit der Dialogfluss-Spezifikation eingelesen und interpretiert werden. In Abhängigkeit von dem Empfänger, den der Controller für ein Ereignis im Modell gefunden hat, kann er eine Aktion aufrufen, die Anfrage an eine Maske weiterleiten oder Dialogmodule aufrufen oder

terminieren. Letztere Operationen werden auf einem Modul-Stack ausgeführt, der für jeden Benutzer die verschachtelten Module speichert, die seinen aktuellen Zustand im Dialogsystem ausmachen.

Dieser zentralisierte Dialogsteuerungsansatz hat drei Vorteile gegenüber der zuvor diskutierten zentralisierten Architektur: Erstens ermöglicht die strikte Trennung zwischen Anwendungslogik, Oberflächengestaltung, Dialogfluss-Spezifikation und Dialogsteuerungslogik ein hohes Maß an Flexibilität für die Komponenten aller vier Schichten. Zweitens können aufgrund dieser sauberen Trennung geräteunabhängige Anwendungen mit minimaler Redundanz implementiert werden: Nur die Dialogfluss-Spezifikation und Dialogmasken müssen für die verschiedenen Präsentationskanäle angepasst werden, während die Anwendungslogik nur einmal geräteunabhängig implementiert werden muss und die Dialogsteuerungslogik vom Framework bereit gestellt wird. Schließlich hat die Dialogsteuerungslogik Kenntnis vom gesamten Dialogfluss jedes Kanals und kann so auch komplexe Dialogkonstrukte steuern.

Das Framework wurde unter Verwendung der Java 2 Enterprise Edition realisiert. Um eine Anwendung mit ihm zu realisieren, müssen Entwickler lediglich einfache Java-Klassen für die Aktionen und JavaServer Pages für die Dialogmasken implementieren sowie DFSL-Dokumente mit der Dialogfluss-Spezifikation generieren. Während alle diese Artefakte anwendungsspezifisch sind, kann das Framework als „Black Box“ unverändert wieder verwendet werden.

4 Verwandte Arbeiten

Andere Ansätze, die explizit zur Modellierung der Navigation in Hypertext-Systemen entwickelt wurden, konzentrierten sich oft nur auf daten-intensive Informationssysteme, jedoch nicht auf interaktions-intensive Anwendungen: Die Sprache WebML [CFB00] ermöglicht z.B. durch Verwendung einer abstrakten XML-Sprache für ihr „Presentation Model” die geräteunabhängige Modellierung des Layouts und Erscheinungsbilds von Webseiten, stellt jedoch keine komplexen Dialogkonstrukte oder einen Mechanismus für die Erweiterung generischer (d.h. auf allen Kanälen identischen) Dialogmodule mit kanalspezifischen Fragmenten zur Verfügung. Die Web Composition Language [GBGS98] konzentriert sich auf die Spezifikation der Dialogmaskeninhalte.

Neuere, häufig auf Statecharts [Har87] basierende Ansätze bieten umfassendere Unterstützung für interaktions-intensive Anwendungen: So ermöglichen z.B. StateWebCharts [WBFP04] die Modellierung dynamischer Web-Anwendungen. Schewe et al. [ST01] verfolgen einen formalen Ansatz zur Modellierung von Interaktionen und Medien-Objekten, der die Spezifikation gerätespezifischer Varianten der Medien-Objekte in Abhängigkeit von den Fähigkeiten der Präsentationskanäle erlaubt. Das von Graunke et al. [GFKF03] vorgeschlagene formale Modell für Web-Interaktion hilft schließlich bei der Identifikation und Behandlung unerwarteter Situationen im Dialogfluss (z.B. client-seitiges Backtracking). Diese Ansätze bieten jedoch kaum Mittel zur nichtredundanten Spezifikation von geräteunabhängigen Dialogflüssen für verschiedene Kanäle.

Einen Fortschritt gegenüber dem bereits zuvor diskutierten Struts-Framework [Apa] stellt die JavaServer Faces-Spezifikation dar [Sun]. Dieser Ansatz ermöglicht unter anderem die von der Anwendungs- und Präsentationslogik getrennte Spezifikation des Dialogflusses in sog. Navigationsregeln. Die Ausdruckskraft dieser Regeln bleibt allerdings hinter der von DFN und DFSL zurück, da mit ihnen nur flache Dialogabläufe, jedoch keine verschachtelten Dialogmodule oder komplexere Dialogkonstrukte spezifiziert werden können.

Mit den Herausforderungen einer präsentationskanal-unabhängigen Dialogsteuerung beschäftigen sich auch das Projekt Sisl (Several Interfaces, Single Logic) [BCD00] sowie die Renderer-Independent Markup Language (RIML) [ZLH04]. In diesen Ansätzen sammeln zwischen Client und Server geschaltete Softwarekomponenten die Datenfragmente, die von den auf mehrere Seiten aufgebrochenen Teilmasken eingehen. Im Gegensatz dazu arbeiten wir an einer Erweiterung des existierenden Dialogsteuerungs-Frameworks, die es erlauben soll, die entstehenden Mikro-Dialogflüsse unmittelbar zu steuern. Eine Reihe von Ansätzen (z.B. von Ceri et al. [CDMN04]) verlagert schließlich die Verantwortung für die gerätespezifische Anpassung von Inhalten auf den Client. Wir verfolgen diesen Ansatz jedoch nicht, da unser Ziel ist, dem Idealkonzept des Thin Client möglichst nahe zu kommen.

Wir haben uns gegen die Verwendung einer generischen Notation wie Statecharts zur Modellierung von Dialogflüssen entschieden, da wir die DFN mit möglichst konstruktiver statt lediglich deskriptiver Ausdruckskraft ausstatten wollten, um Entwicklern zu ermöglichen, komplexe Dialogkonstrukte intuitiv durch Nutzung entsprechender Elemente verwenden zu können, ohne seine Details mit den Mitteln einer generischen Notation ausformulieren zu müssen. Infolgedessen enthält die Notation einige Elemente, die zunächst aus formaler Sicht nicht zwingend erforderlich erscheinen, die praktische Anwendbarkeit des Frameworks auf typische Entwicklungsprobleme jedoch steigern sollten.

5 Ausblick und Fazit

Ungeachtet des hier befürworteten pragmatischen Ansatzes zur Dialogfluss-Spezifikation und -Steuerung muss als solide Basis für künftige Forschungsarbeiten natürlich eine formale Semantik der Dialogfluss-Notation definiert werden (zusätzlich zur bereits durch die Framework-Implementierung geschaffene operationale Semantik). Dies kann dadurch erreicht werden, dass gezeigt wird, dass alle DFN-Konstrukte auch mit Hilfe eines generischen Formalismus wie z.B. Petrinetzen ausgedrückt werden können, auch wenn dieser für den praktischen Einsatz zu unhandlich wäre. Die Definition einer solchen formalen Basis ist derzeit in Arbeit.

Eine praktische Herausforderung, die noch zu lösen ist, stellt die sinnvolle Handhabung client-seitiger nichtlinearer Navigation wie z.B. Betätigung des Back-Buttons eines Browsers dar. Aufgrund ihrer häufigen Verwendung durch Benutzer sollte sie als normales Interaktionsmuster betrachtet werden, das genauso sauber zu behandeln ist wie normale Klicks auf Links. Backtracking stellt jedoch ein Problem für jede zustandsbasierte Dialogsteuerungslogik dar, da es Ereignisse (d.h. Auslöser von Zustandsübergängen) wiederholt, die

im aktuellen Zustand des Dialogflusses nicht definiert sind. Abgesehen von der technischen Frage, wie dadurch erneut erzeugte Ereignisse zu identifizieren sind, liegt die eigentliche Herausforderung darin, sie nicht nur technisch fehlerfrei, sondern auch im Anwendungskontext semantisch korrekt zu behandeln (indem der Benutzer auf eine frühere Seite zurückgeführt wird, eine Aktion rückgängig gemacht wird o.ä.). Besonders interessant wird dieses Problem durch die Präsenz verschachtelter Dialogstrukturen, die u.U. keine beliebigen Rücksprünge erlauben. Die Entwicklung von Lösungen für diese Problematik ist ein Kerngebiet unserer künftigen Forschung.

Um die praktische Anwendbarkeit der Notation und des Frameworks zu untersuchen, haben wir bereits ein Touristinformations- und Reiseplanungssystem als Web-Anwendung mittlerer Komplexität entwickelt, die alle Dialogsteuerungsfeatures nutzt und Präsentationskanäle für Desktop-PCs, PDAs und Mobiltelefone bietet. Dieses Projekt bestätigte die Praktikabilität des hier vorgeschlagenen Ansatzes und brachte insbesondere insofern ermutigende Ergebnisse, als die Realisierung der mobilen Präsentationskanäle aufgrund der vollständigen und unveränderten Wiederverwendung der Anwendungslogik in einem Bruchteil der für den Desktop-Kanal verwendeten Zeit möglich war.

Wir sind bestrebt, weitere Erfahrungen mit größeren Projekten zu sammeln, um die Praxistauglichkeit von Notation und Framework sowie ihren Einfluss auf den Software-Entwicklungsprozess und die Software-Qualität zu evaluieren. Während verwandte Methoden dazu tendieren, die web-basierte Oberfläche mehr oder weniger direkt aus einem etablierten Datenmodell abzuleiten, macht die DFN keine Annahmen über das zugrunde liegende Datenmodell, sondern beschreibt ausschließlich die Interaktion der Benutzer mit dem System. Sie sollte daher einen dialoggetriebenen Entwicklungsprozess unterstützen, der die ISO-Dialoggrundsätze der Aufgabenangemessenheit und Erwartungskonformität von Anfang an betont: Da die Interaktionsmuster aus den Benutzeranforderungen abgeleitet werden, können sie den Entwurf der Anwendungslogik und des Datenmodells mitformen, statt von ihnen diktiert zu werden. Unsere durch die Evaluation noch zu überprüfende Hypothese ist, dass dieser Ansatz sowohl die Effizienz des Entwicklungsprozesses als auch die Benutzerfreundlichkeit von Web-Anwendungen steigert.

Literatur

[Apa] Apache Software Foundation. The Apache Struts Web Application Framework. http://struts.apache.org.

[BCD00] T. Ball, C. Colby und P. Danielsen. Sisl: Several Interfaces, Single Logic. *International Journal of Speech Technology*, 3(2):91–106, 2000.

[BG03] M. Book und V. Gruhn. A Dialog Control Framework for Hypertext-based Applications. In H. Lin und H.D. Ehrich, Hrsg., *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, Seiten 170–177. IEEE Computer Society Press, 2003.

[BGGW02] M. Butler, F. Giannetti, R. Gimson und T. Wiley. Device Independence and the Web. *IEEE Computing*, 6(5):81–86, Sep–Oct 2002.

[CDMN04] Stefano Ceri, Peter Dolog, Maristella Matera und Wolfgang Nejdl. Model-Driven Design of Web Applications with Client-Side Adaptation. *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004), Lecture Notes on Computer Science*, 3140:201–214, 2004.

[CFB00] Stefano Ceri, Piero Fraternali und Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33:137–157, 2000.

[GBGS98] M. Gaedke, M. Beigl, H.W. Gellersen und C. Segor. Web Content Delivery to Heterogeneous Mobile Platforms. *Advances in Database Technologies, Lecture Notes in Computer Science*, 1552, 1998.

[GFKF03] P.T. Graunke, R.B. Findler, S. Krishnamurthi und M. Felleisen. Modeling Web interactions. *Proceedings of the 12th European Symposium on Programming, Lecture Notes on Computer Science*, 2618, 2003.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[iso96] Ergonomic requirements for office work with visual display terminals (VDTs) — Part 10: Dialogue principles. Bericht ISO 9241-10, International Organization for Standardization, 1996.

[JHE99] J. Jing, A. Helal und A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(6):117–157, Jun 1999.

[Kra88] G.E. Krasner. A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[RFPG96] J. Rice, A. Farquhar, P. Piernot und T. Gruber. Using the web instead of a window system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '96)*, 1996.

[ST01] K.-D. Schewe und B. Thalheim. Modeling Interaction and Media Objects. *Proceedings of the 5th International Conference on Applications of Natural Language to Information Systems, Lecture Notes in Computer Science*, 1959:313–324, 2001.

[Sun] Sun Microsystems. JavaServer Faces. http://java.sun.com/j2ee/javaserverfaces/.

[WBFP04] M. Winckler, E. Barboni, C. Farenc und P. Palanque. SWCEditor: A Model-based Tool for Interactive Modelling of Web Navigation. In *Proceedings of the 5th International Conference on Computer-Aided Design of User Interfaces (CADUI 2004)*, Seiten 55–66. Kluwer Academics, 2004.

[ZLH04] T. Ziegert, M. Lauff und L. Heuser. Device Independent Web Applications — The Author Once – Display Everywhere Approach. *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004), Lecture Notes in Computer Science*, 3140:244–255, 2004.